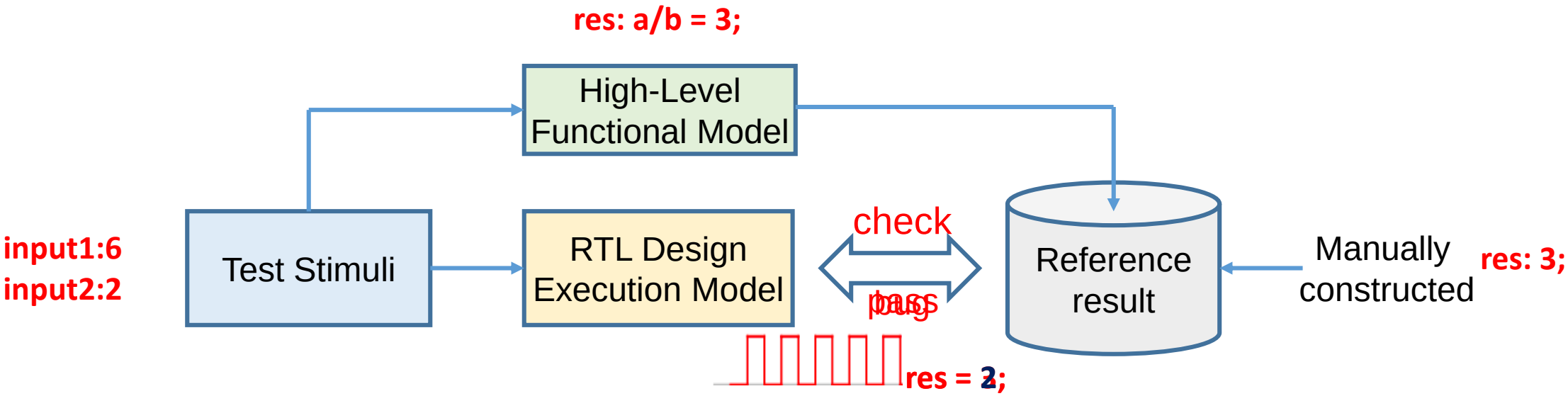
Generalizing and Automating Tandem Simulation: Connecting High-level and RTL Simulation Models

Yue Xing, Aarti Gupta, Sharad Malik
Princeton University
yuex@princeton.edu

This work is supported by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA. This research is also funded in part by NSF award number 1628926, and the DARPA POSH Program.

General Simulation-based Testing

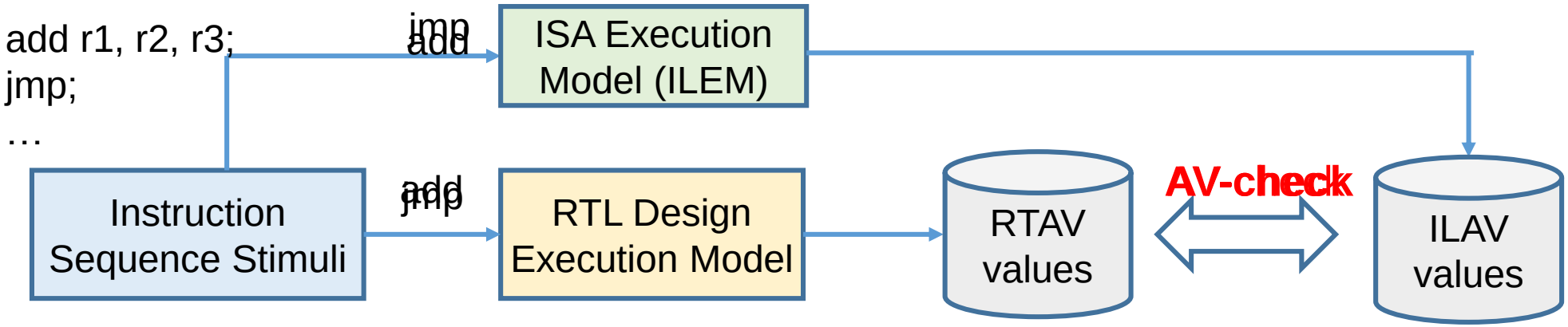
- Activate the RTL Execution Model (RTEM) with test stimuli (a sequence of RTL inputs)
- After the simulation, check if the RTL signal/register value matches the reference result



Deficiency: need to run and analyze the full test regardless of when bug is triggered – instead, should stop when bug is triggered

Tandem Simulation for Processors

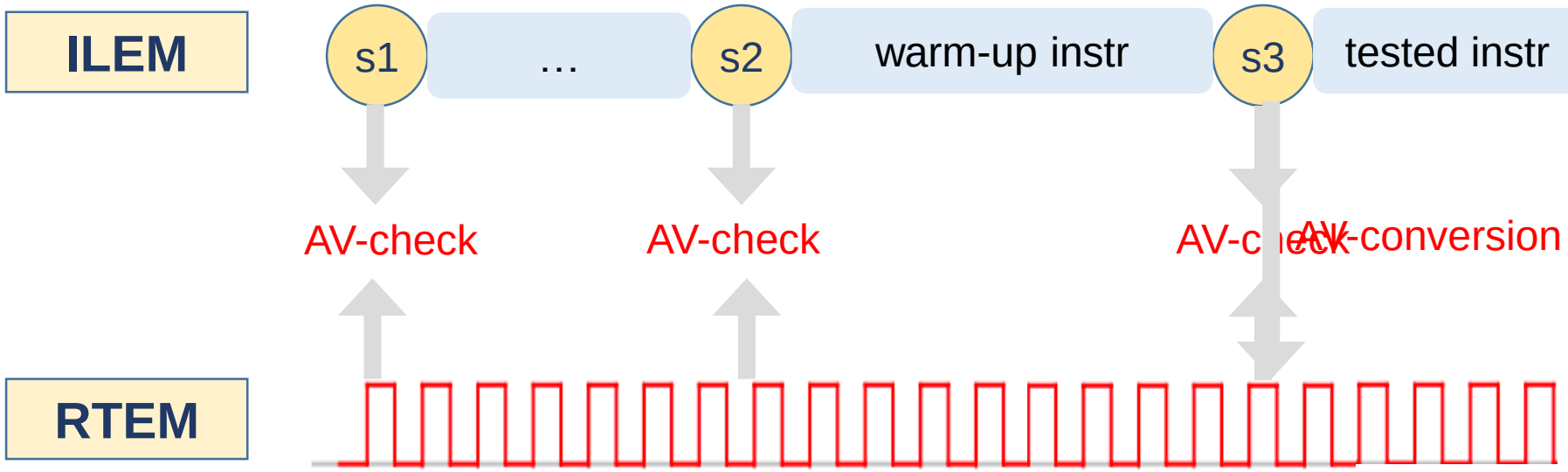
- Instruction-by-instruction simulation [HCM+ DATE14] ;Cross-level simulation [CM VLSI13]
- Activate the Instruction-Level Execution Model (ILEM) and RTEM instruction-by-instruction
- After simulating each instruction, check if the RTEM Architectural Variables (RTAV) match ILEM Architectural Variables (ILAV)
  - ILAV: pc, r0-r31, special regs, …



Identify bugs right at the instruction that causes AV deviations

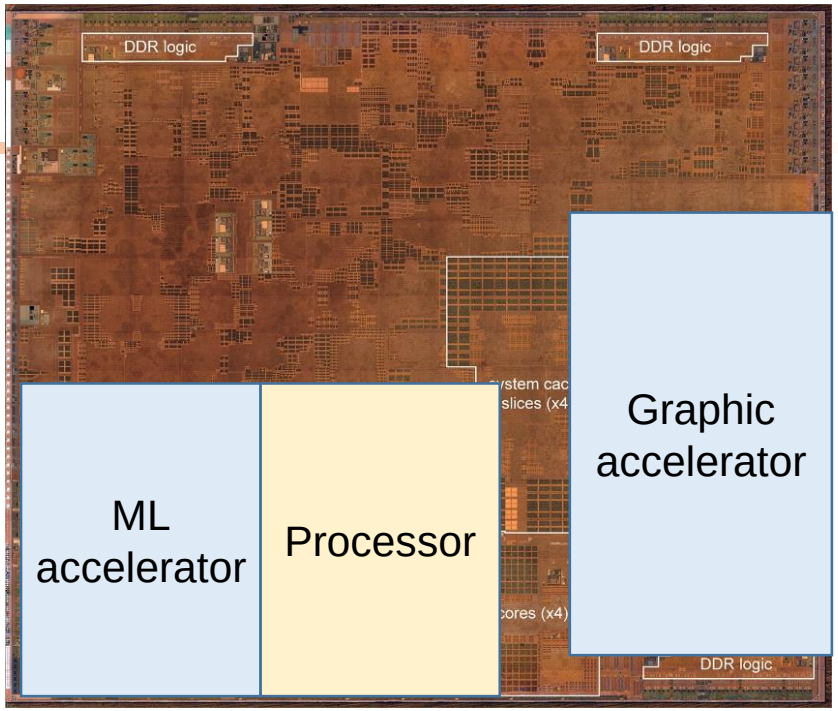
Tandem Simulation Scenarios

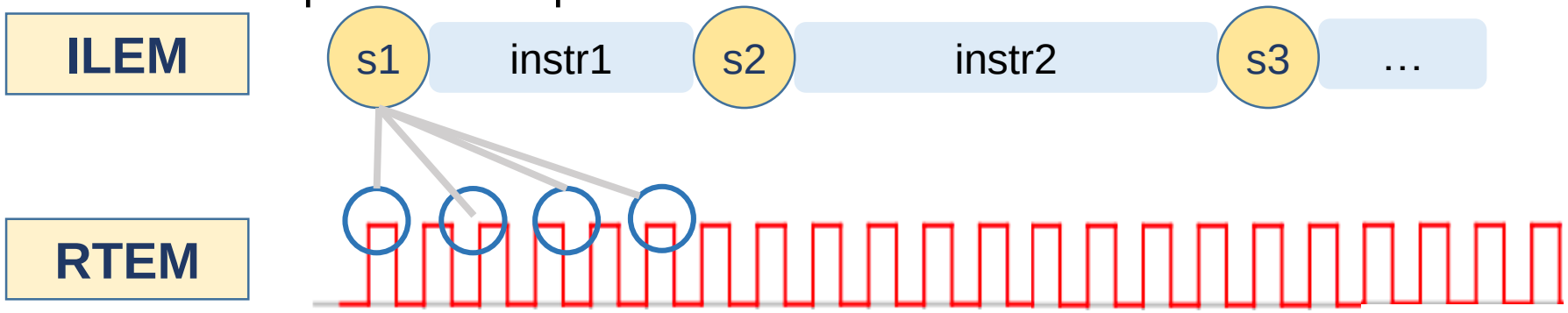
- Scenario 1: Instruction-by-instruction AV-check
  - State deviation indicates bugs – can be debugged with nearby instructions
- Scenario 2: Checking at checkpoints (e.g. every 2 instructions)
  - Invoke checking less often to avoid the overhead of AV-check
- Scenario 3: Jump-starting RTEM Simulation
  - Reduce simulation time by leveraging “warm-up” test phase to only the ILEM



Limitations of Current Work

- Lack of generalization
  - Only applied to processor, which has the instruction-level model -- ISA
- Lack of automation
  - Need manual input to
    1) Identify corresponding AVs for comparison
    2) Identify the end of instructions to trigger comparison/swap





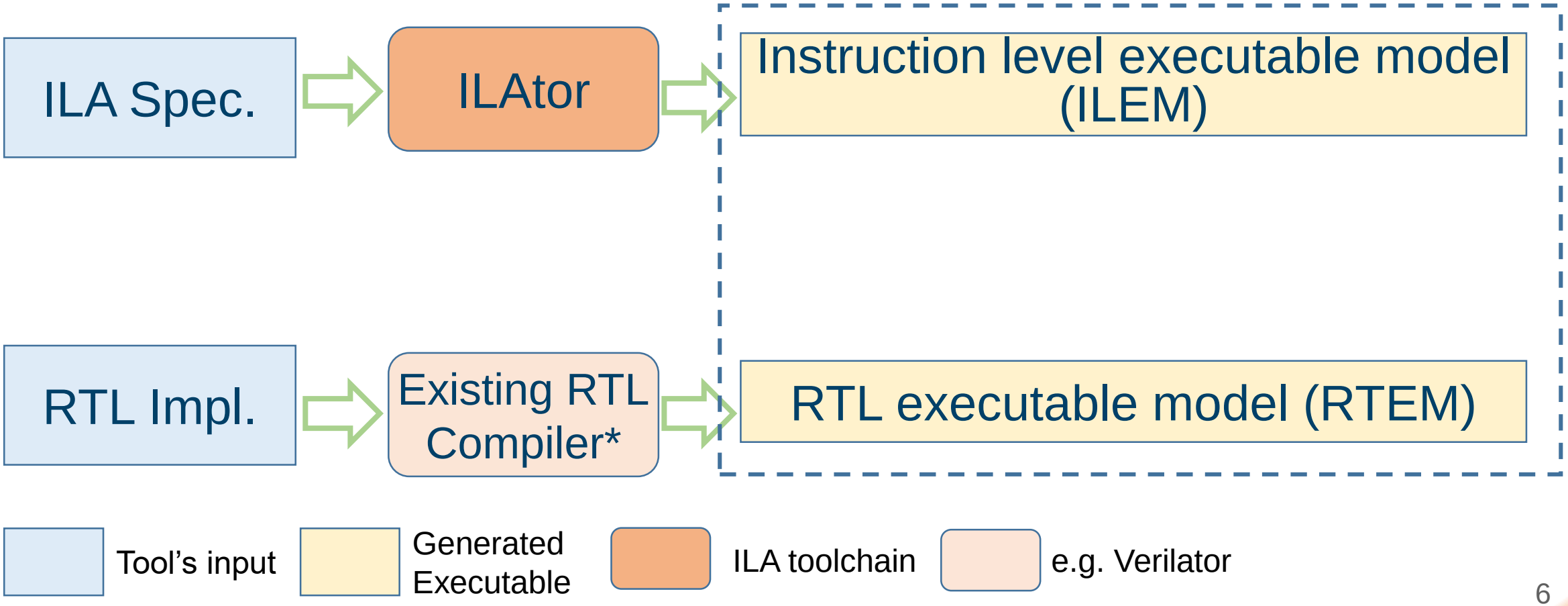
Leverage Instruction-Level Abstraction (ILA) and refinement map to address the gaps

Outline

- Introduction
- **Methodology**
- Addressing Challenges
- Experiments

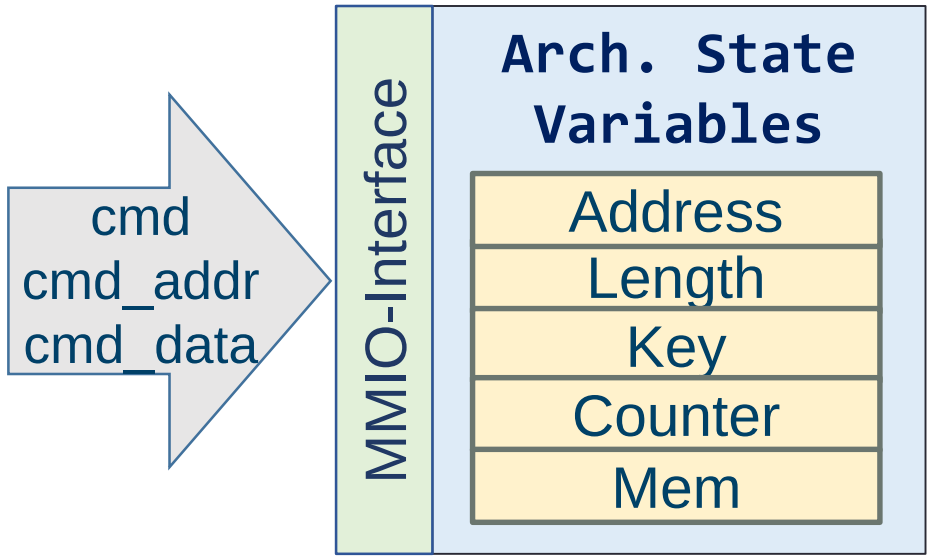
Methodology Overview

- Leverage ILA to generalize tandem simulation to other SoC components – MMIO accelerators



Instruction-Level Abstraction (ILA)

- ILA[1]: architectural-level abstraction to uniformly model processors and accelerators
  - A set of architecture state variables (AVs) – variables that are persistent between instructions
  - A set of instructions – interface commands that update the AVs



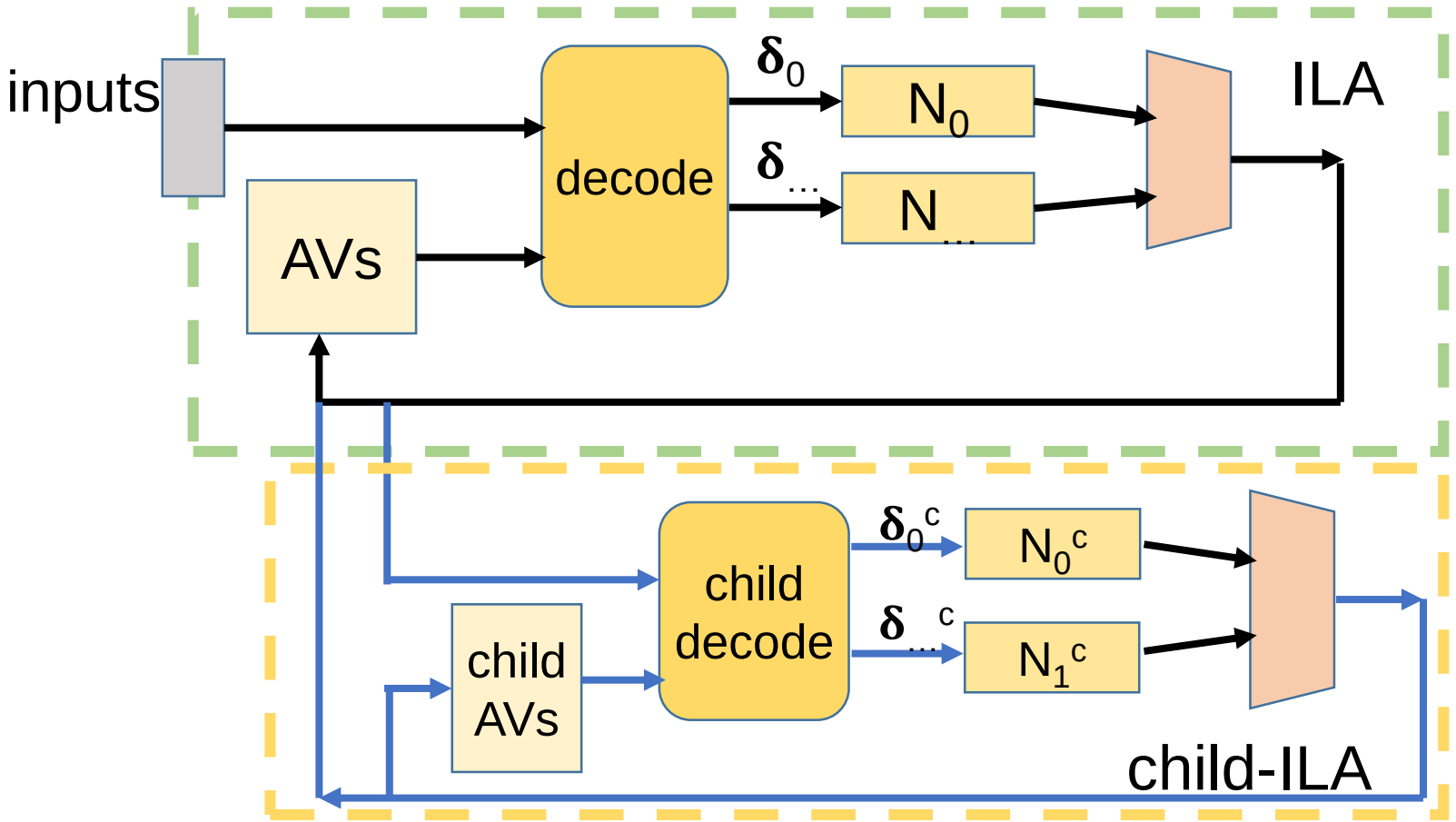
Instructions (I)		
I_0	set key	$\delta_0 := (cmd\_addr == 0xff10) \&\& (cmd == 2)$ $N_0(key) := cmd\_data$
$I_{...}$	…	…
I_4	start encrypt	• I_0^c Load data block • I_1^c Encrypt • I_2^c Store data block

ILA of an AES accelerator design

[1]B.-Y. Huang, et al, "Instruction-Level Abstraction (ILA): A Uniform Specification for System-on-Chip (SoC) Verification," TODAES, 2018

ILAtor

- Synthesize an executable model from an ILA specification



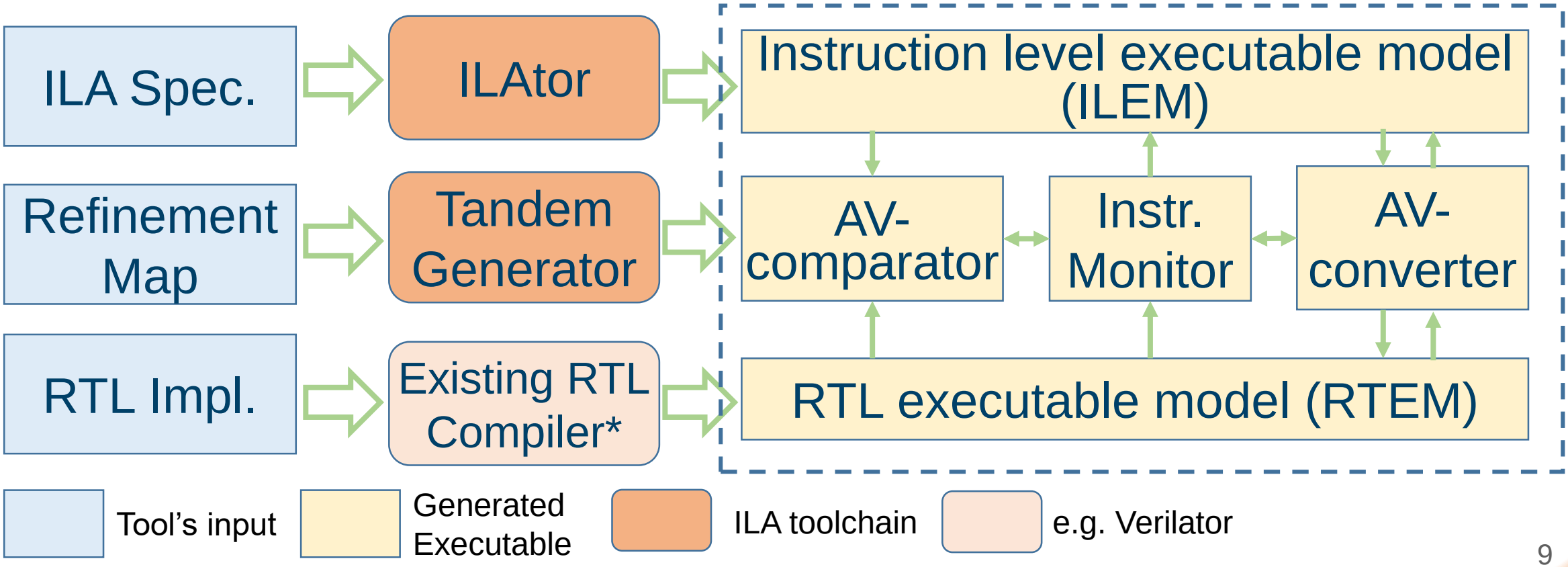
ILA execution model

```
kernel() {
  // Parent-ILA
  if (δ0)
    instr0.update();
  ...
  // Child-ILAs
  do {
    if (δ0c)
      child_i0.update();
    ...
  } until (no more
                child instr)
}
```

ILEM template

Methodology Overview

- Leverage ILA to generalize tandem simulation to other SoC components – MMIO accelerators
- Leverage refinement map for automation



Refinement Map

- Commonly used in formal verification
- AV map: **what** to check
  - Mapping between ILAVs and RTAVs
- Instruction map: **When** to check
  - The time/condition when each instruction starts and finishes in RTL

AV map	
AES-ILA	**AES-RTL**
key	top.aes_key.out
length	top.aes_length.out
addr	top.aes_addr.out
status	top.status_reg
counter	top.aes_counter.out

instruction map		
Instruction	**start condition**	**finish condition**
set key	decode	1 cycle
start encrypt	decode	status == 0

Automatically Generated Tandem Simulation

- AV-map → AV-Comparator and AV-Converter
- Instruction map → Instruction Monitor

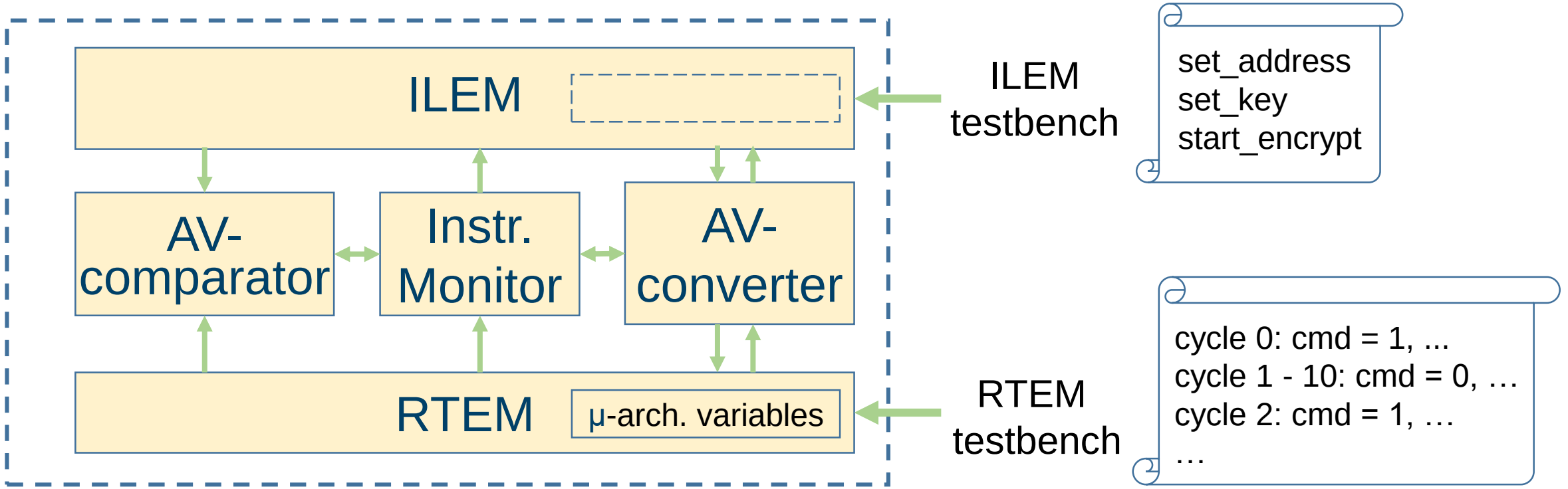
AV map	
AES-ILA	**AES-RTL**
key	top.aes_key.out
length	top.aes_length.out
addr	top.aes_addr.out
status	top.status_reg
counter	top.aes_counter.out

instruction map		
Instruction	**start condition**	**finish condition**
set key	decode	1 cycle
start encrypt	decode	status == 0

```
AV_Comparator() {
  if (ilem.key != rtem.aes_key.out or …)
    error
}
AV_Converter() {
  rtem.aes_key.out = ilem.key;
  …
}
Instr_Monitor() {
  exe_queue <- any_start(rtem);
  finish_instr <- any_finish(exe_queue)
  if (finish_instr)
    AV_Comparator();
}
```

Challenges

- ILEM vs. RTEM testbenches
- AV swapping and micro-architectural variables

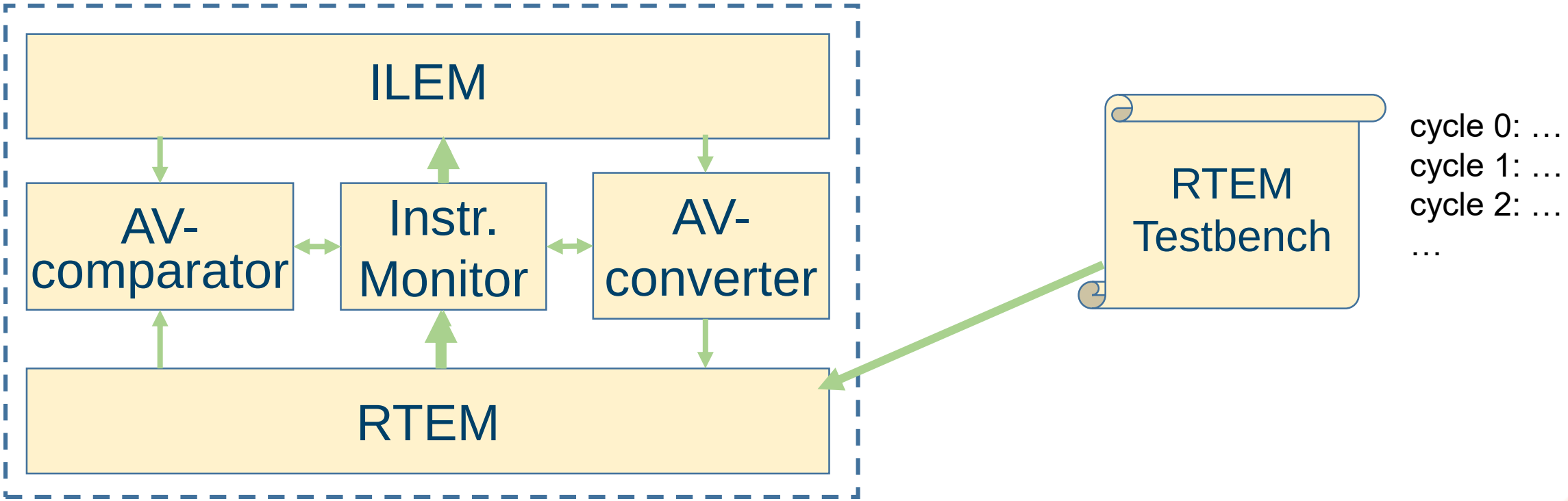


Outline

- Introduction
- Methodology
- **Addressing Challenges**
- Experiments

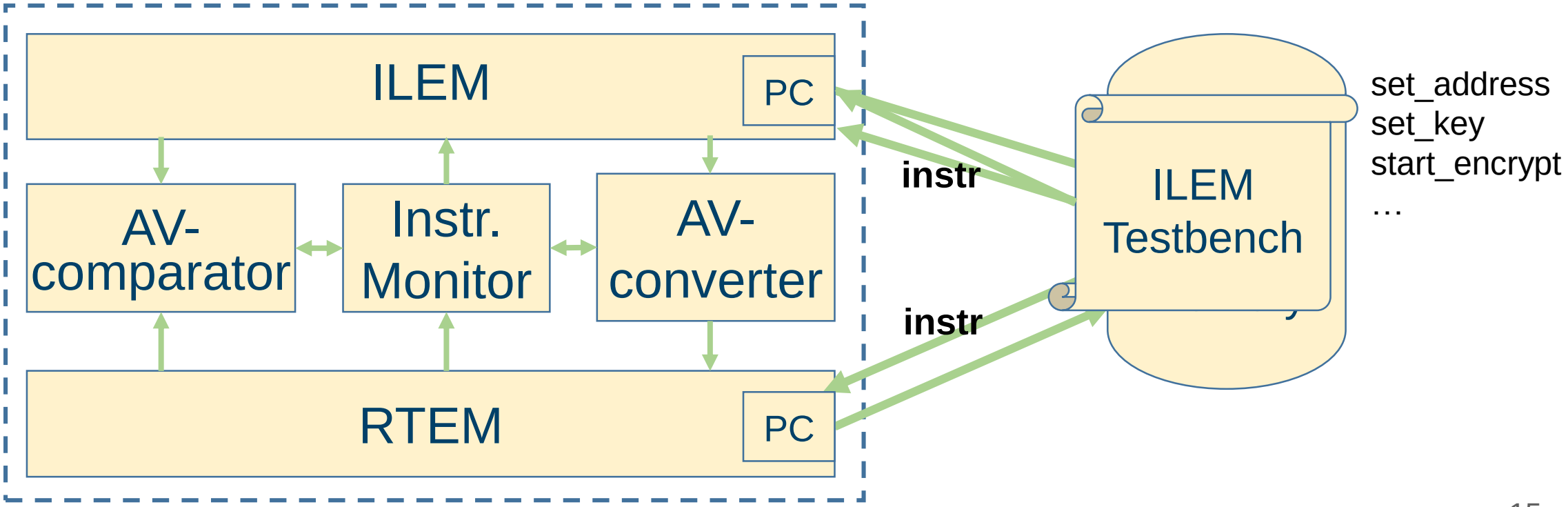
Tandem Simulation with A Single Testbench

- Single RTEM testbench
  - An RTEM testbench is a cycle-by-cycle description of inputs
  - Instr. monitor captures when an instruction starts/ends and then trigger ILEM to run the corresponding instruction



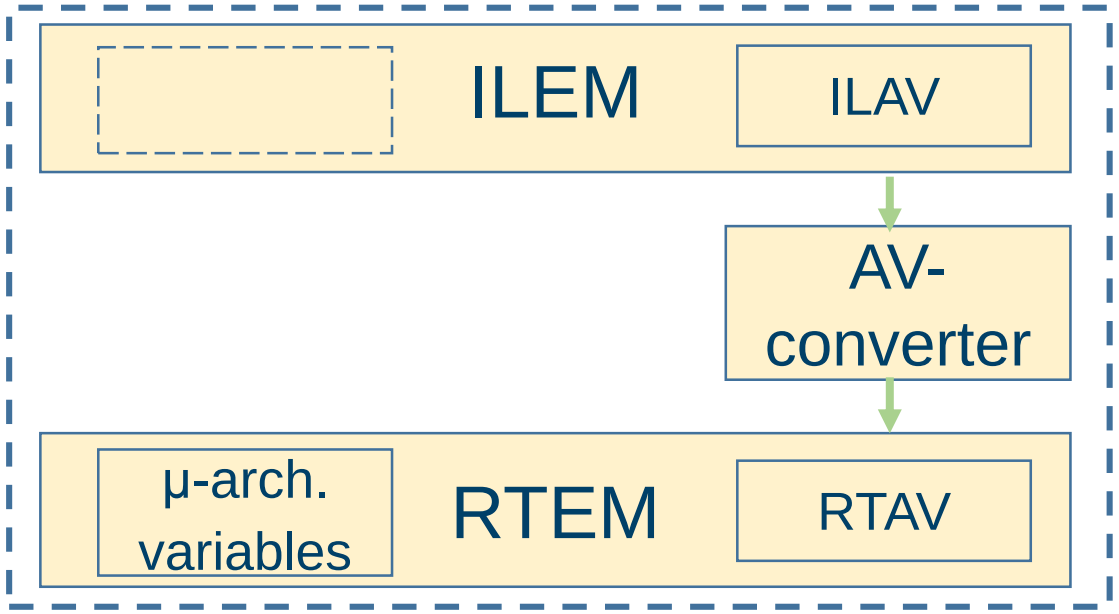
Tandem Simulation with A Single Testbench

- Single ILEM testbench
  - An ILEM testbench is a sequence of instruction inputs
  - Similar to processor, we store the ILEM testbench in an instruction memory
  - An auxiliary state “program counter” is added to both ILEM and RTEM



Micro-architectural Variables in AV-Converting

- Similar to processor, initialize micro-arch. variables through “cold start”
  - Apply a “code start” sequence to reset the whole RTEM
  - Convert the ILAV to RTAV
- Specify the “code start” information in the refinement map
  - Resetting input sequence
  - # of cycles to hold converted RTAV



“cold start” map	
pre-swap cycle/sequence	e.g. (1) reset (m1 cycles)
	e.g. (2) reset = [1, 0, …], global start = [0, 1, 0…]
conversion hold cycle	e.g. m2 cycles

Outline

- Introduction
- Methodology
- Addressing Challenges
- **Experiments**

Experiments

- Seven Case Studies
  - Accelerators: FlexNLP, AES-block, AES-round, GaussianBlur (GB)
  - Processors: Pico, Piccolo, Rocket
- Experiment Overview
  - Runtime evaluation (breakdown) of each tandem simulation component
  - Simulation speedup with jump-starting
  - Improvement in bug detection time

[FlexNLP]: T. Tambe, et al., “Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference,” in DAC, 2020
[AES]: H. Hsing, “OpenCores.org: Tiny AES,” 2014, [Online]. https://opencores.org/project/tiny_aes
[GaussianBlur]: J. Pu, et al. “Programming Heterogeneous Systems from an Image Processing DSL,” TACO, 2017
[pico]: C. Wolf, “PicoRV32,” 2020, [Online] https://github.com/cliffordwolf/picorv32
[piccolo]: Bluespec, Inc., “BlueSpec RISC-V designs,” 2020, [Online] https://github.com/bluespec,
[rocket]: K. Asanovic, et al., “The rocket chip generator,” Tech. Rep. UCB/EECS-2016-17

Experiments – runtime evaluation

- Seven case studies – 3 scenarios

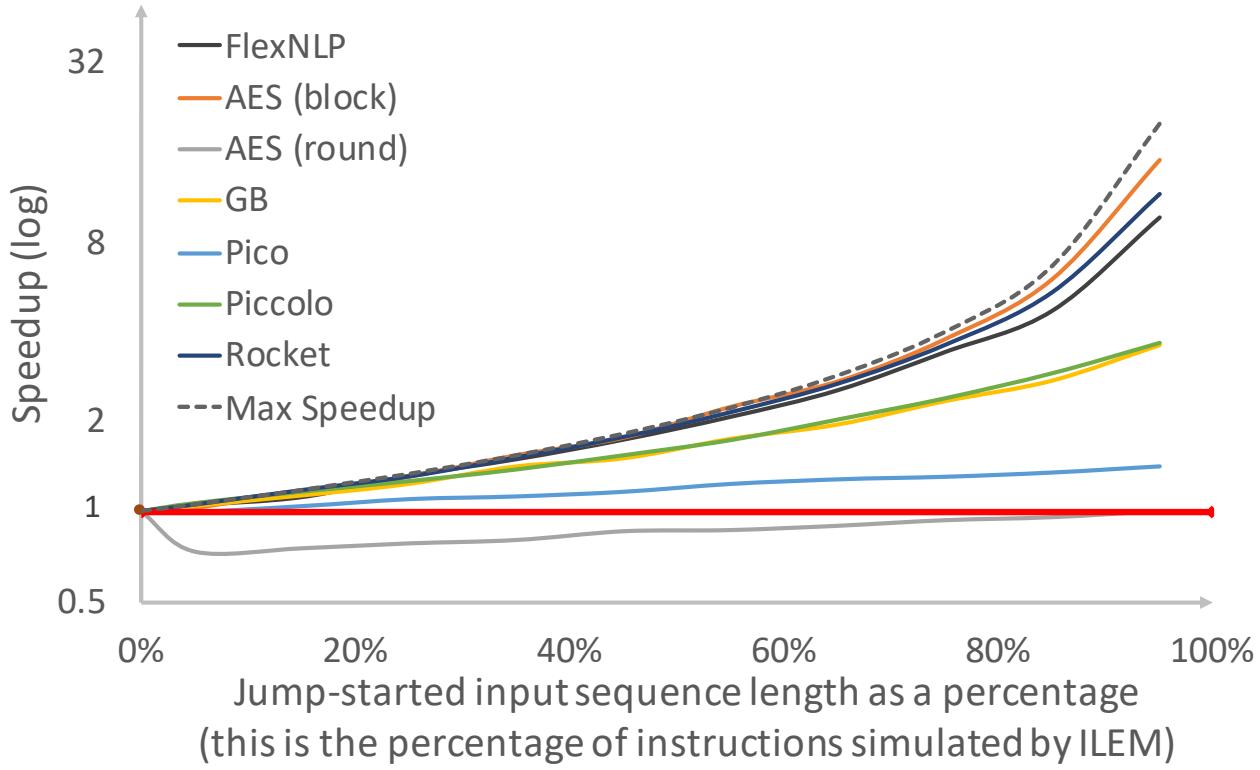
	Design Statistics				Simulation Time Statistics				
Design	**ILA** (LoC)	**# of Arch. state bits**	**RTL** (LoC)	**Ref-map** (LoC)	**RTEM** (us/instr)	**ILEM** (us/instr)	**S1[x]** (us/instr)	**S2[x]** (us/instr)	**S3[x]** (us)
FlexNLP	5807	5008	18338	459	2999	262	17.6	0.083	16694
AES (block)	236	298	1078	73	387	7.3	0.25	0.033	74.1
AES (round)	236	298	321	62	7.49	7.3	0.64	0.2	80.9
GB	285	621	11375 (1325†)	147	3.3	1.2	0.19	0.066	14
Pico RISC-V	584	1056	2014	208	0.97	0.29	0.084	0.024	0.4
Piccolo RISC-V	584	1056	6063 (4122†)	223	4.5	0.3	0.26	0.022	789
Rocket RISC-V	584	1056	13468 (3856†)	213	101	0.29	0.85	0.029	652

†Lines of code for HLS design

[x] S1 – Scenario 1, AV-check overhead; S2 – Scenario 2, checkpoint overhead; S3 – Scenario 3, AV-convert overhead

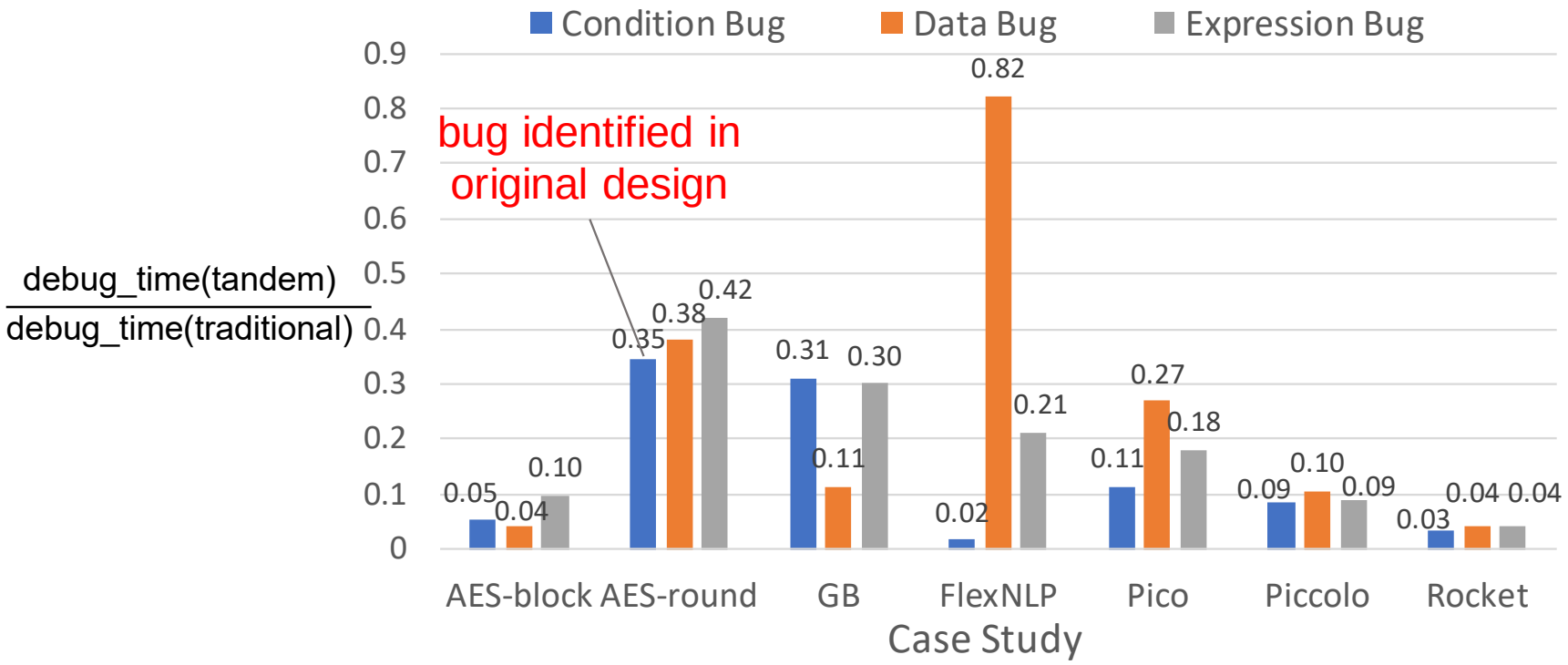
Experiments – Simulation Speedup

- Simulation speedup with jump-starting
  - Divide practical testcases into “warm-up” phase and important phase
  - “Warm-up” phase only runs on ILEM; important phase runs on CLEM



Experiments – bug detection time

- Two debugging strategies
  - Traditional conformance testing: run the test to the end and compare results
  - Tandem simulation: run the test and apply AV-Check at the end of each instruction



Experiments Summary

- Automation of tandem simulation for processors and accelerators
  - Leverage ILA model and refinement map
- Negligible runtime overhead
- Significant simulation speedup with jump-starting
- Early bug detection

Conclusion

- An automatic flow for generalizing tandem simulation to accelerators
  - Leverage ILA to generalize tandem simulation to other SoC components
  - Leverage refinement map to automate the checks/jump-starting Address challenges in testbenches and AV-swapping
  - Efficacy demonstrated through multiple case studies
- Tool open-sourced in ILAng github repo: https://github.com/PrincetonUniversity/ILAng
  Case studies: https://github.com/yuex1994/ASPDAC-tandem